B.TECH. PROJECT REPORT

ON

CLOUD-BASED TOOL FOR RELIABILITY ESTIMATION

BY

Durbha Aditya

Soham Kapileshwar

DISCIPLINE OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

May 2022

CLOUD-BASED TOOL FOR RELIABILITY ESTIMATION

A PROJECT REPORT

Submitted in partial fulfilment of the

requirements for the award of the degrees

of

BACHELOR OF TECHNOLOGY

In

MECHANICAL ENGINEERING

Submitted by:
Durbha Aditya
Soham Kapileshwar

Guided by:

Dr Bhupesh Kumar Lad

INDIAN INSTITUTE OF TECHNOLOGY INDORE
May 2022

CANDIDATE'S DECLARATION

We hereby declare that the project entitled "**CLOUD-BASED TOOL FOR RELIABILITY ESTIMATION**" submitted in partial fulfilment for the award of the degree of Bachelor of Technology in Mechanical Engineering completed under the supervision of **Dr Bhupesh Kumar Lad, Professor, IIT Indore** is an authentic work.

Further, we declare that we have not submitted this work for the award of any other degree elsewhere.

Soham Kapileshwar 26/05/22

Durbha Aditya 26/05/22

Signature and name of the student(s) with date

CERTIFICATE by BTP Guide(s)

It is certified that the above statement made by the students is correct to the best of my/our knowledge.

25/5/2022

Bhupesh K Lad

Professor

Signature of BTP Guide(s) with dates and their designation

Preface

This report on “Cloud Based Web Tool for Reliability Estimation” is prepared under the guidance of Dr Bhupesh Kumar Lad, Professor, IIT Indore.

Through this report, we have explained the development of the Cloud-Based Web Tool and tried to cover every aspect of its functioning. We have also explained why this web tool is economical and useful for SMEs and how it tackles the disadvantages of more comprehensive software such as BlockSim.

We have tried to the best of our abilities and knowledge to explain the content in a simplified manner by using illustrative diagrams. We have also explained how to use the Web Tool in a step by step manner.

Durbha Aditya, Soham Kapileshwar

B.Tech. IV Year

Discipline of Mechanical Engineering

IIT Indore

Acknowledgements

We wish to thank Dr Bhupesh Kumar Lad, Professor, for his kind support and valuable guidance.

It is with his help and support we were able to complete the design and technical report.

Without his support, this report would not have been possible.

Durbha Aditya, Soham Kapileshwar

B.Tech. IV Year

Discipline of Mechanical Engineering

IIT Indore

Abstract

Reliability is defined as the probability for a system to function properly without failure for a specified period of time. Softwares such as BlockSim, and Weibull++ provide a comprehensive platform for system reliability, availability, and related analyses. However, these platforms have certain disadvantages such as -

1. Needs to be installed on a local device with certain CPU requirements.

2. Incurs heavy charges, irrespective of the frequency of use.

To solve these problems our project aims at creating a cloud-based platform. This platform is an interactive online tool that allows users to create analytical or simulation reliability block diagrams and perform analysis on the same. The analytical approach estimates reliability using theoretical formulas of standard distributions for failure. The simulation approach involves simulating a system for a certain period of time and iterations by randomly generating failure and repair times based on its distribution. The scope of this project can be extended to include various other functionalities to provide more insightful results and plots, allowing small enterprises to use it on a “pay-per-use” model, hence providing a more feasible solution.

Table of Contents

List of Figures

Chapter 1 - Introduction

1.1 Background

Reliability and maintainability are relatively recent engineering fields. Increased system complexity and sophistication, public knowledge of and demand for product quality, new rules and regulations controlling product liability, and so on have all fostered their growth.

In life-cycle costing, cost-benefit analysis, operational capability evaluations, repair and facility resourcing, inventory and spare parts requirement estimations, replacement selections, and the implementation of preventive maintenance programmes, reliability and maintainability are key functions.

1.2 Reliability Engineering

Reliability engineering is an engineering discipline that involves applying scientific knowledge to a component, product, facility, or process in order to guarantee that it performs its intended function in a specific environment for the needed time period without failure.

The probability of a system (component) functioning throughout a time period t is defined as reliability. Let us define the continuous variable T (random) as the system's time to failure in order to represent this connection analytically (component).
Then reliability can be expressed as-

R(t) = The probability of survival of the component till time t.

Or

$$R(t) = \Pr\{T \geq t\}$$

R(t) is the probability that the failure time is more than or equal to t , for a known value of t.

Let us define $$F(t) = 1 - R(t) = \Pr\{T < t\}$$

$$F(0) = 0$$

and $$\lim_{t \to \infty} F(t) = 1$$

then F(t) is the probability that a failure occurs before time t.

We define a third function called the **probability density function** which can be written as

$$f(t) = \frac{dF(t)}{dt} = -\frac{dR(t)}{dt}$$

Thus R(t),F(t) and f(t) can be together written as

R(t) = P (T > t) = 1-F(t) = 1- ∫ f(t). dt

where,

t = time for calculation

T = time of failure

f(t) = probability density function

1.3 Analytical vs Simulation Approach

An Analytical Approach to Reliability means estimating reliability using theoretical formulas of standard distributions. For example, the reliability of a component at a particular instant, following Weibull distribution, can be estimated by using,

$$R(t) = \exp\left[-\int_0^t \frac{\beta}{\theta}\left(\frac{t'}{\theta}\right)^{\beta-1} dt'\right]$$
$$= e^{-(t/\theta)^{\beta}}$$

Simulation Approach of Reliability means we can simulate the provided system for a length of time and a number of iterations using principles from reliability theory to derive an estimate of probable outputs in the form of those parameters.

Simulation can be done by doing a couple of things for every happening event:
1) Pseudo Random Number generation.
2) Inverting the Failure function.
When these two are coupled, they may imitate a component for a length of time by producing failure and repair times based on a random number produced from its failure distribution.

1.4 Applications of Reliability Engineering

Consider the following example to understand the application of Reliability.

The B. A. Miller Company manufactures small motors for household appliances such washing machines, dryers, refrigerators, and vacuum cleaners. One of the motors it produced had an exceptionally high failure rate, with 43 failures in the first 1000 units produced. Several of these blunders were seen by the appliance manufacturer during final product testing. The bearing case looked to be spinning in its seat when it was examined. The sealed ball bearings, on the other hand, looked to be in good condition. These failures might have been caused by defective components, a design flaw, a faulty material or a manufacturing (tolerance) issue.A testing study revealed that motors manufactured at the conclusion of a production run failed at a higher rate than those produced at the beginning of the run.

The MTTF of the first 300 units produced is 3773.5 operating hours (7545/2), and the MTTF of the last 200 units produced is 360.7 operating hours (4328/12). As a result, it was considered that the manufacturing process was spiralling out of control and that design tolerances were failing to be reached. As a result, the firm increased its focus on its quality control programme in order to avoid early motor failures.

In decreasing order of priority, the following are the goals of reliability engineering:

- To use engineering knowledge and specialised procedures to avoid or lessen the possibility or frequency of failure.
- Identifying and correcting the reasons for failures that occur despite best attempts to avoid them.
- To figure out how to deal with failures that do occur if the reasons have not been addressed.

- Methods for predicting the anticipated dependability of new designs and analysing reliability data must be used.

1.5 Pre-existing modules used

Backend:

- **Django**: Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design.
- **Django-rest-framework**: Django REST framework is a powerful and flexible toolkit for building Web APIs.
- **Numpy**: NumPy is **a Python library used for working with arrays**. It also has functions for working in the domain of linear algebra, fourier transform, and matrices.

Frontend:

- **Excalidraw**: **Excalidraw** is a virtual collaborative whiteboard tool that lets you easily sketch diagrams that have a hand-drawn feel to them.
- **Axios**: Promise based HTTP client for the browser and **node**.js.
- **Material-UI**:Material-UI is an open-source project that features React components that implement Google Material Design.

Chapter 2 - Methodology

2.1 Fundamentals of Reliability Engineering

2.1.1. Weibull Distribution

The Weibull probability distribution is quite useful in reliability. Both growing and decreasing failure rates may be modelled using the Weibull failure distribution. It is characterised by a hazard rate function of the form,

$$\lambda(t) = \frac{\beta}{\theta}\left(\frac{t}{\theta}\right)^{\beta-1} \qquad \theta > 0,\ \beta > 0,\ t \geq 0$$

$$R(t) = \exp\left[-\int_0^t \frac{\beta}{\theta}\left(\frac{t'}{\theta}\right)^{\beta-1} dt'\right]$$

$$= e^{-(t/\theta)^{\beta}}$$

$$f(t) = -\frac{dR(t)}{dt} = \frac{\beta}{\theta}\left(\frac{t}{\theta}\right)^{\beta-1} e^{-(t/\theta)^{\beta}}$$

Beta is referred to as the shape parameter. Theta is a scale parameter that influences both the mean and the spread, or dispersion, of the distribution. The parameter theta is also called the characteristic life, and it has units identical to those of the failure time, T.

2.1.2. Effect of Theta and Beta on Weibull Probability Density Function

For beta < 1, the PDF is similar in shape to the exponential, and for large values of beta, the PDF is somewhat symmetrical, like the normal distribution. For 1 < beta < 3, the density function is skewed. When beta = 1, lambda(t) is a constant, and the distribution is identical to the exponential with lambda = 1/theta.

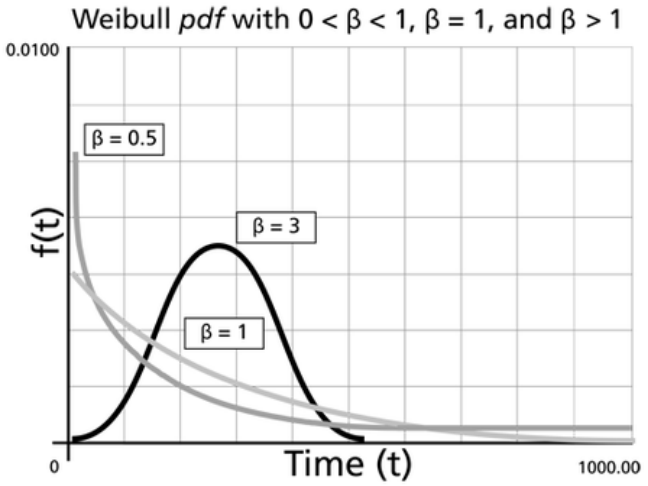


Fig: 1

As theta increases, the reliability increases at a given point in time.

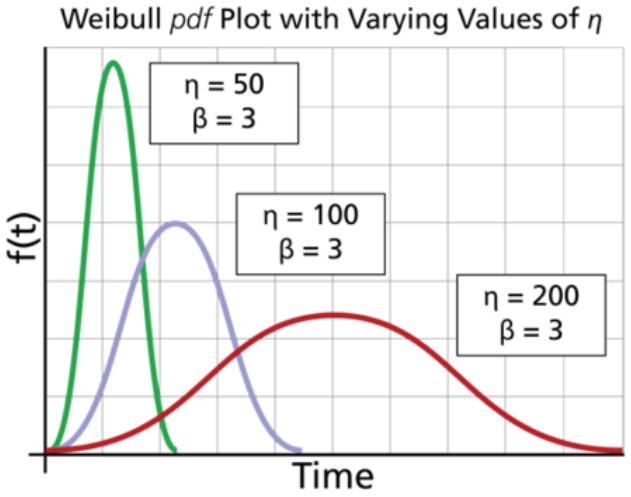


Fig: 2

2.1.3. Normal Distribution

The normal distribution has successfully been utilised to represent fatigue and wear out phenomena. It is also useful in assessing lognormal probabilities due to its link with the lognormal distribution. The formula for the PDF is,

$$f(t) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\frac{(t-\mu)^2}{\sigma^2}\right] \qquad -\infty < t < \infty$$

The normal is not a true reliability distribution since the random variable ranges from minus infinity to plus infinity. However, for most observed values of mean and standard deviation, the probability that the random variable will take on negative values is negligible, and the normal can therefore be a reasonable approximation to a failure process.

The reliability function for this distribution is determined from,

$$R(t) = \int_t^{\infty} \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\frac{(t' - \mu)^2}{\sigma^2}\right] dt'$$

2.2 The Concept of Simulation

2.2.1. Monte Carlo Simulation

The Monte Carlo simulation approach is used in system reliability analysis to create random failure periods from each component's failure distribution. Simulating system functioning and experimentally estimating reliability values for a series of time values yields the total system dependability. Simulation has become a highly common analytical method because to the introduction of computers. Simulation is simple to use and can create outcomes that are challenging to answer analytically. Simulation methods, on the other hand, have certain downsides, not the least of which is that the findings are dependent on the number of simulations, resulting in a lack of reproducibility.

Other shortcomings include the inability to model systems with static components (i.e., components whose dependability does not fluctuate over time) and the inability to use most reliability optimization and allocation approaches.

2.2.2. Inverse Transform Sampling

Inverse transform sampling, also known as inverse transformation function sampling, is a fundamental method of generating pseudo-random numbers from any probability distribution with a cumulative density function.

We may use the following method to produce random numbers based on a particular distribution because the 'Random' module in Python, like any other language or computer platform, can generate uniform pseudo-random integers in a specific range:

1. Generate a random number ‘u’ from the standard uniform distribution in the interval [0,1].
2. Find the inverse transform function of F-1 (x) of the Cumulative Density Function of given distribution.
3. Compute X = F-1(u). The computed random variable has the required distribution.

Chapter 3 - Maintenance and System Configurations

3.1 The Concept of Maintenance

3.1.1. Introduction

The possibility of executing a successful repair operation within a certain time frame is described as maintainability. In other words, maintainability measures how readily and fast a system can be brought back up after a failure. This is related to system reliability analysis, except the random variable of concern in maintainability analysis is time-to-repair rather than time-to-failure. For example, if a component is reported to have a 90% maintainability for one hour, it means that there is a 90% likelihood that it will be repaired in that time. When you combine system maintainability with system reliability analysis, you may get a lot of important information regarding overall performance that will help you make design decisions.

3.1.2. Types of Maintenance

Corrective Maintenance	Preventive Maintenance
Corrective Maintenance is done only after a component fails.	Preventive Maintenance can be done even if the component is in working condition.
It is an easier and more direct method since it does not require any preparation to prevent asset failure.	It is more complicated than Corrective Maintenance as it requires failure prevention planning.
Because particular device failure causes greater system damage, it may be more expensive than Preventive Maintenance.	It is costly, but it prevents assets from failing.
It can greatly impact the system as certain damages to particular components can result in severe losses.	It saves production losses by lowering the probability of failure.
It is randomly conducted depending on when the failure occurs.	It is conducted at regular intervals as components should be examined regularly.

In this project, we are assuming the corrective as well as preventive maintenance follows the

Gaussian Distribution only.

Types of Preventive Maintenance

- **Age-Based Maintenance** : Age based maintenance is done when the component/system surpasses a certain age and is likely to undergo failure.
- **Time-Based Maintenance**: Time based maintenance is done every certain duration irrespective of the component's/system's condition.

3.2 System Configurations

The components of a system are linked together by a configuration that determines the state system as a function of individual component states.

System configurations used in this project are:-

1. Series configuration

   The system will fail if any of the components fails. The reliabilities of different components combine to form the system's reliability.
   R(system) = R(BLOCK-1) * R(BLOCK-2) * R(BLOCK-3)

   The series configuration of components is schematically represented as follows:

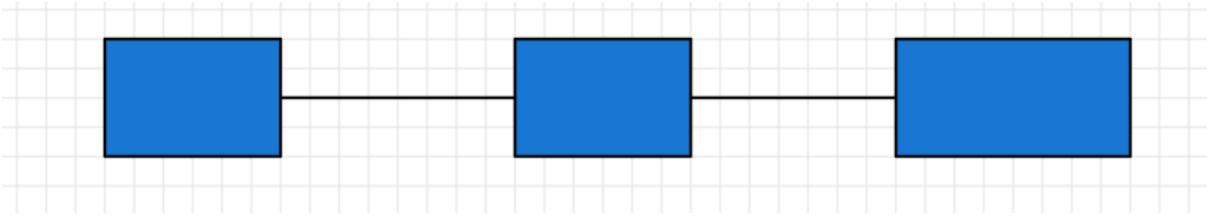


Fig: 3

2. Parallel Configuration

When at least one of the components is active, the system is in parallel configuration. The product of the failure probabilities of the constituent components is the system's failure probability.

F(system) = F((BLOCK-1) * F(BLOCK-2) * F(BLOCK-3)
i.e. R(system) = 1 – (1-R(BLOCK-1)) (1-R(BLOCK-2)) (1-R(BLOCK-3))

The parallel configuration of components is schematically represented as follows:

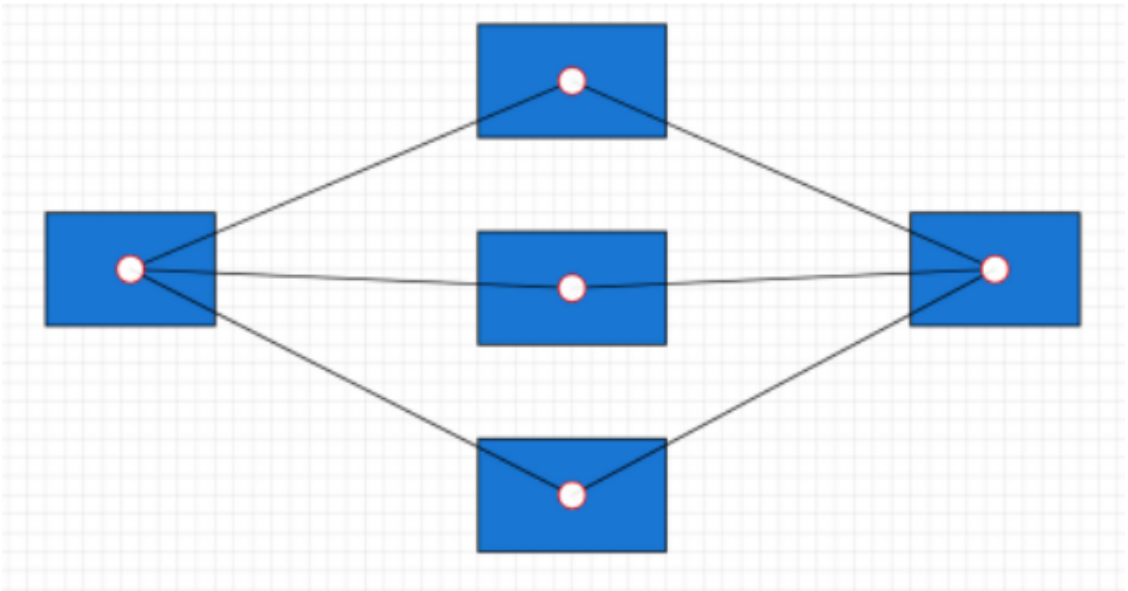


Fig: 4

Chapter 4 - Overview of a Cloud-Based WebTool

4.1 Problem Statement

Reliability calculations are necessary for enterprises as it helps in the maintenance and cost-effectiveness of systems. Softwares such as BlockSim, Weibull++, provide a comprehensive platform for system reliability, availability, maintainability, and related analyses. However, they suffer from many disadvantages.

1. Needs to be installed on your device with certain minimum CPU requirements.
2. They incur heavy charges. Installation costs, especially for an SME will be really expensive.
3. Entire software needs to be purchased even if the requirement is infrequent. They do not follow the "Pay-Per-Use" model.

Thus the solution is:

A cloud-based web tool can be made for systems or RBDs (Reliability Block Diagrams) in order to tackle the disadvantages mentioned in the previously.

1) Cloud-Based so that it doesn't need to be installed on your device.

2) Affordable cost. A "Pay-Per-Use" model would be preferable and would incur minimum cost to the users.

3) Interactive and easy-to-use User Interface.

4) Development of software using open-source languages and frameworks.

4.2 Basics of Web Application Development

4.1.1 Introduction

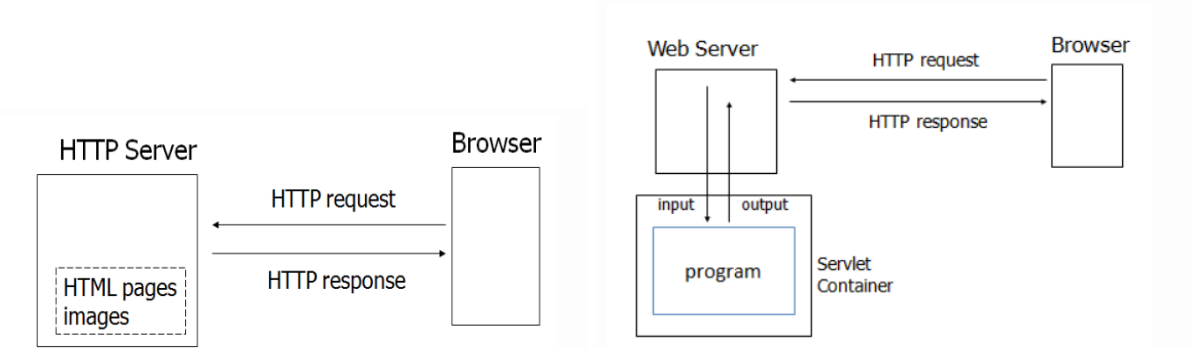
Web application development is the process of developing, designing, testing and deploying a web-based application.

Web apps are interactive pages that operate on a web server and allow for user interaction. A web application is distinguished by the fact that it is hosted on the internet and accessible via a browser. They're also safer, easier to backup, and less expensive than mobile app development.

4.1.1 Static vs Dynamic

Web apps can be basic static web pages or dynamic and interactive web applications.

Static web pages are saved in the web server's file system and show the same information to all visitors. Dynamic pages, on the other hand, are created by a software that generates HTML. This sort of online application gives the user personalised information and allows them to customise the material according to their preferences.



Static Dynamic

Fig: 5

4.1.2 Frontend vs Backend

The front and back ends of a website are developed separately. Front end development is a type of programming that concentrates on the visual aspects of a website or app that a user will interact with (the client side). Back end development, on the other hand, concentrates on the aspect of a website that visitors cannot see (the server side).

Frontend Frameworks: React, Angular

Backend Frameworks: Express, Django

4.3 Introduction to cloud-based

Cloud-based refers to programmes, services, or resources that are made accessible to customers on demand through the Internet from the servers of a cloud computing provider.

Companies usually use cloud-based computing to increase capacity, improve functionality, or introduce new services on demand without having to commit to potentially expensive infrastructure investments or hire/train more in-house support people.

The type of cloud computing we will be dealing with is **Software as a Service (SaaS).**

4.4 Web Hosting

Web hosting is an online service that makes the content of your website available to the public. When you buy a hosting package, you're renting space on a real server where all of your website's files and data will be stored.
Web hosts provide the technology and resources required for the effective and secure running of your website. They are in charge of keeping the server up and running, implementing security regulations, and ensuring that data like as texts, images, and other files are transferred to visitors' browsers effectively.

Common web hosting services include AWS , Hostinger , Heroku.

4.5 Flow Chart of the web-tool

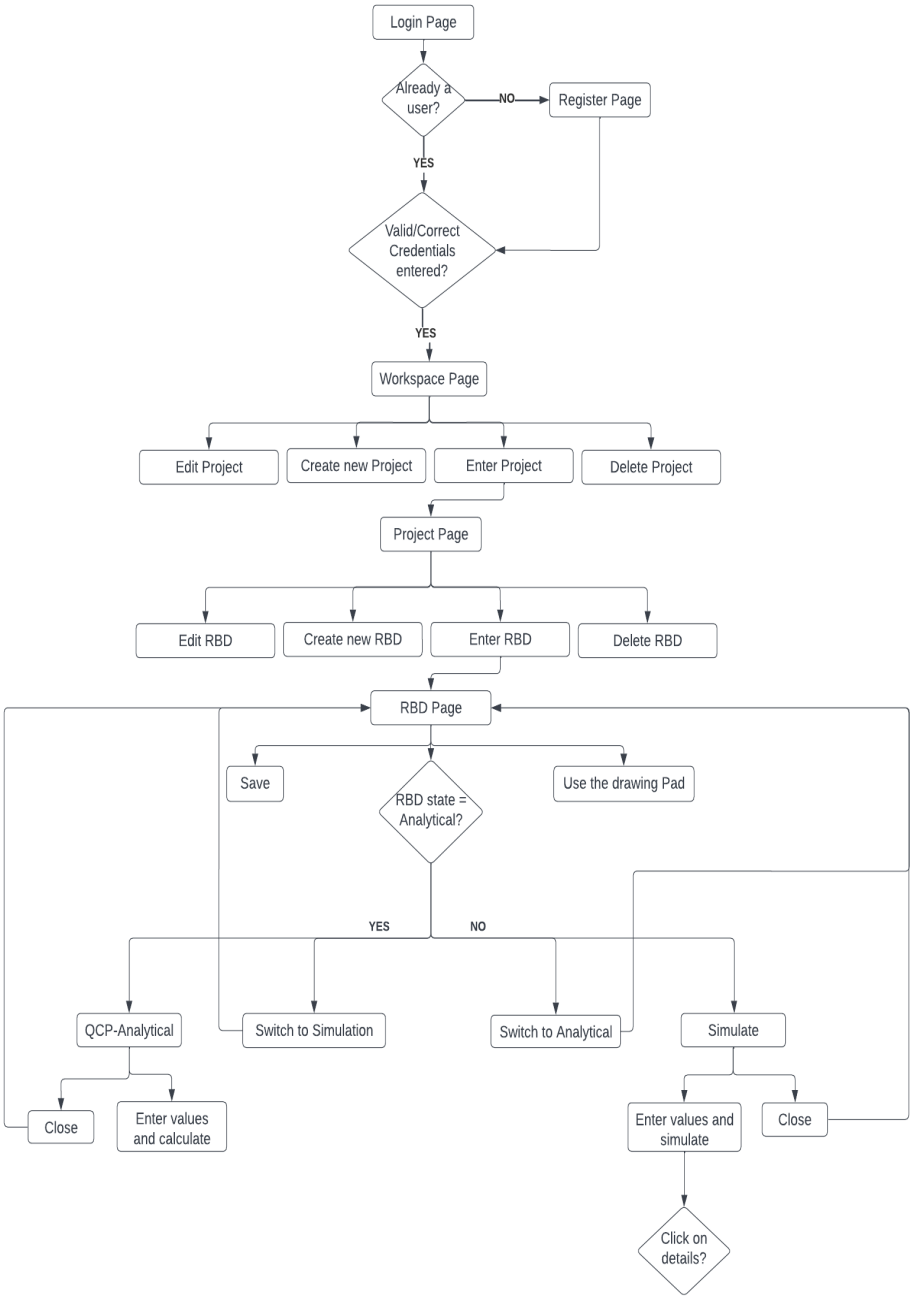


Fig: 6

Chapter 5 - Development of WebTool: Frontend

5.1 Introduction

Front-end web development is **the development of the graphical user interface of a website, through the use of HTML, CSS, and JavaScript, so that users can view and interact with that website**.

The front end of our web tool is built using the 'React Js' library.

REACT:
React is an efficient, declarative, and flexible open-source JavaScript library for building simple, fast, and scalable frontends of web applications. It is component-based and each component has its own state.
React Components are **independent and reusable bits of code**. They serve the same purpose as JavaScript functions, but work in isolation and return HTML. Components come in two types, Class components and Function components.

Through **API calls** we can send/ retrieve information to/from the backend and use it in the frontend.

REDUX:
Redux is an open-source JavaScript library used to manage application state. React uses Redux for building the user interface.

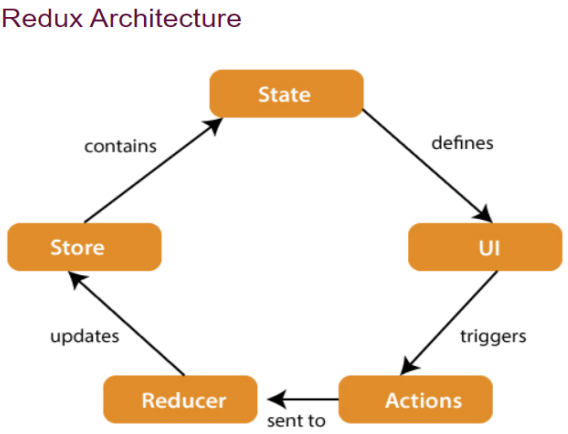


Fig: 7

EXCALIDRAW:

Excalidraw is a virtual collaborative whiteboard tool that lets you easily sketch diagrams that have a hand-drawn feel to them.

NPM:

NPM is an abbreviation for Node Package Manager. It is a Node JavaScript platform package manager. NPM is widely regarded as the world's largest software registry. NPM is used by open-source developers all around the world to publish and distribute their source code.

5.2 URLs

A **URL** (Uniform Resource Locator) is a unique identifier used to locate a resource on the Internet. It is also referred to as a web address.

In a React app, React-Router is **the standard routing library**. This allows us to link URLs to the components/pages.

The URLs we have created in our frontend web tool are:-

1. **"/register"**: Path for the register page
2. **"/"**: Path for login page
3. **"/workspace"**: Path for Workspace Page
4. **"/workspace/:project_name/:project_id"**: Path for Project Page

5. **“/workspace/:project_name/:project_id/:rbd_name/:rbd_id”** : Path for RBD page

(‘:’ is used for variable names in the URL , for example in ‘/:project_name’ , project_name is the variable name)

5.3 Pages

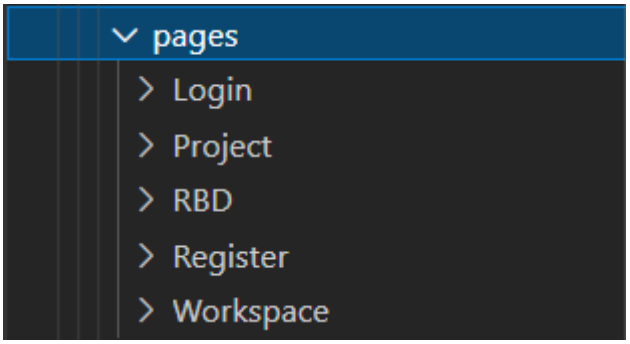


Fig: 8

Based on the URL , the corresponding pages are visible to the user.

The list of Pages in our web tool is given below-

1. Login Page:
   Page which renders the LoginComponent.

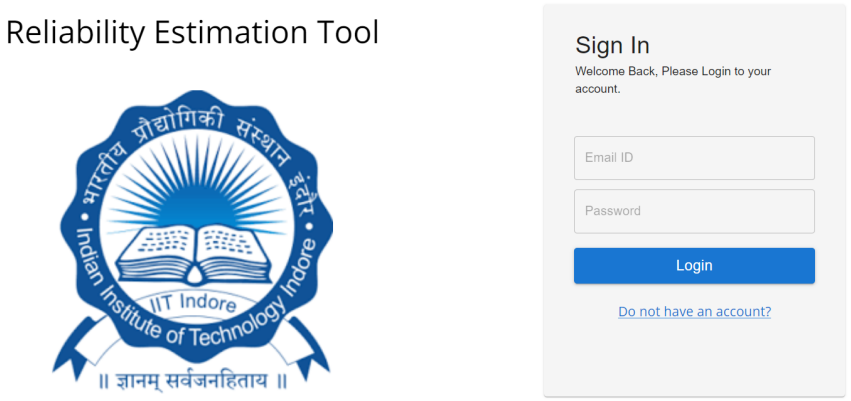


Fig: 9

2. Register Page:

   Page which renders the RegisterComponent.

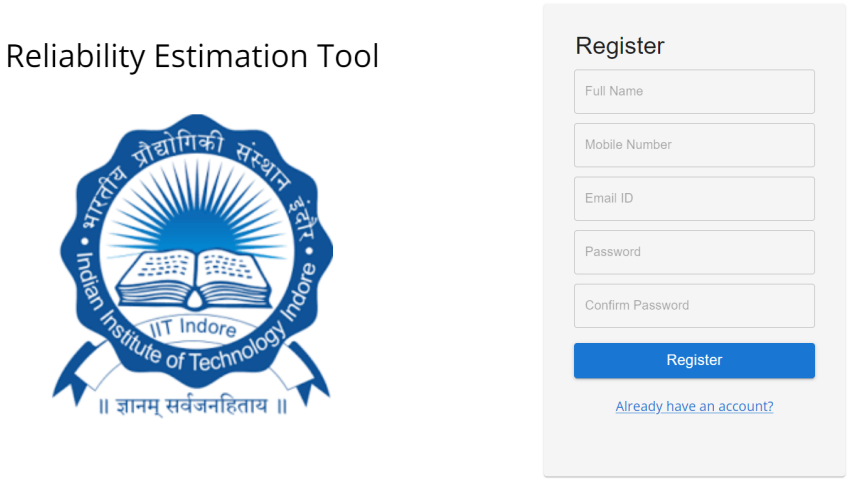


Fig: 10

3. Workspace Page:

   Page which displays the users' existing list of projects from which the user can select and also create/edit/delete projects.

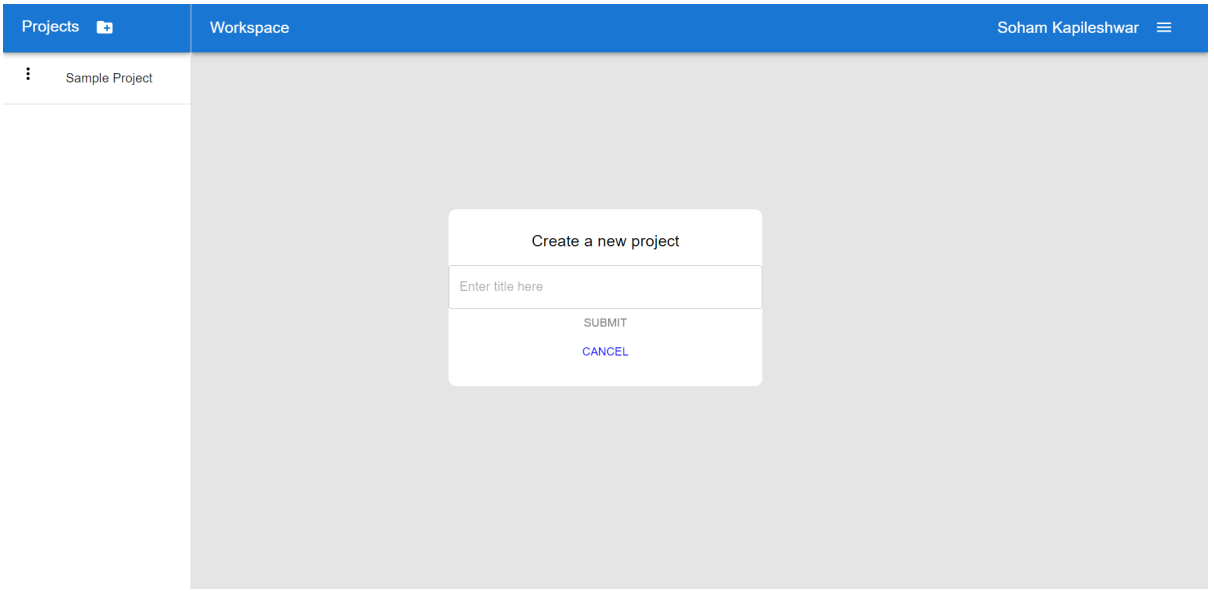


Fig: 11

4. Project Page:

   Page which displays the list of RBDs within the selected project from which the user can select and also create/edit/delete RBDs..

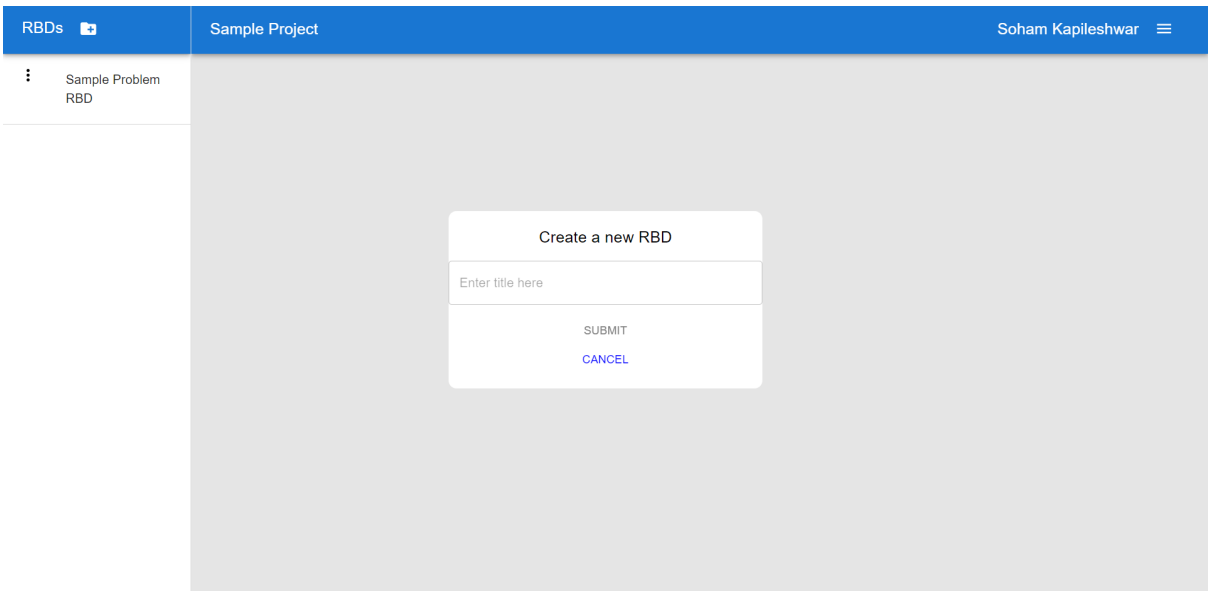


Fig: 12

5. RBD Page:

Page which renders the WorkingPad Component to allow users to draw RBDs, save their progress and perform the required analysis.

SIDEBAR MENU OPTIONS:-

1. **Switch to Simulation/Switch to Analytical** switches the mode and provides MENU options accordingly.
2. **Save** button makes an API call to the backend and saves the current RBD state.

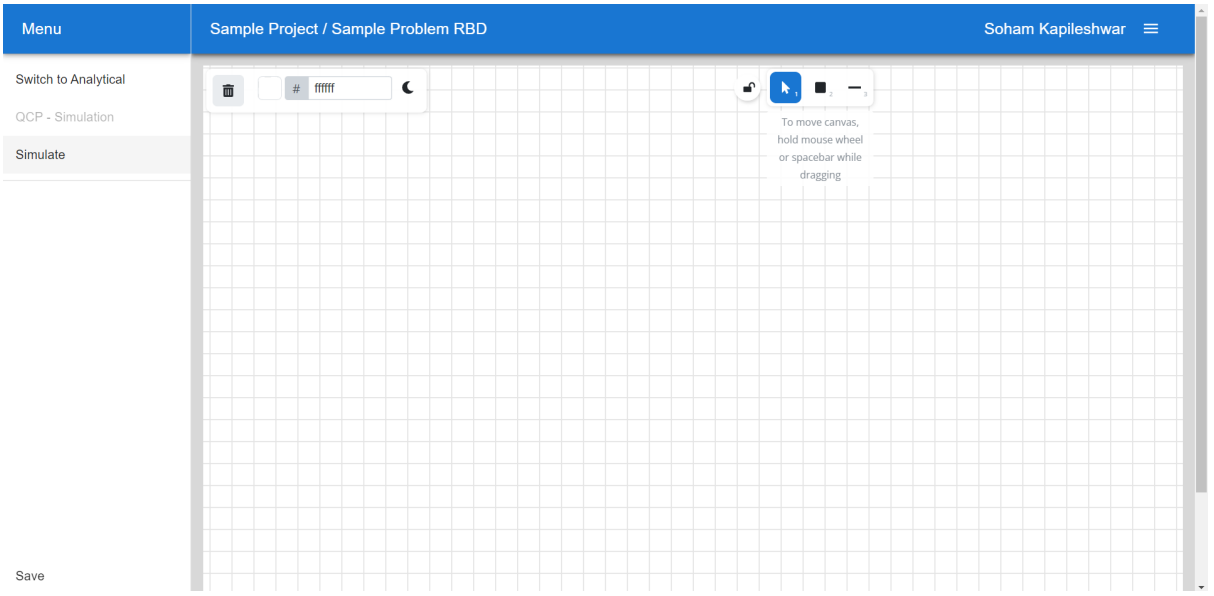


Fig: 13

5.4 Components

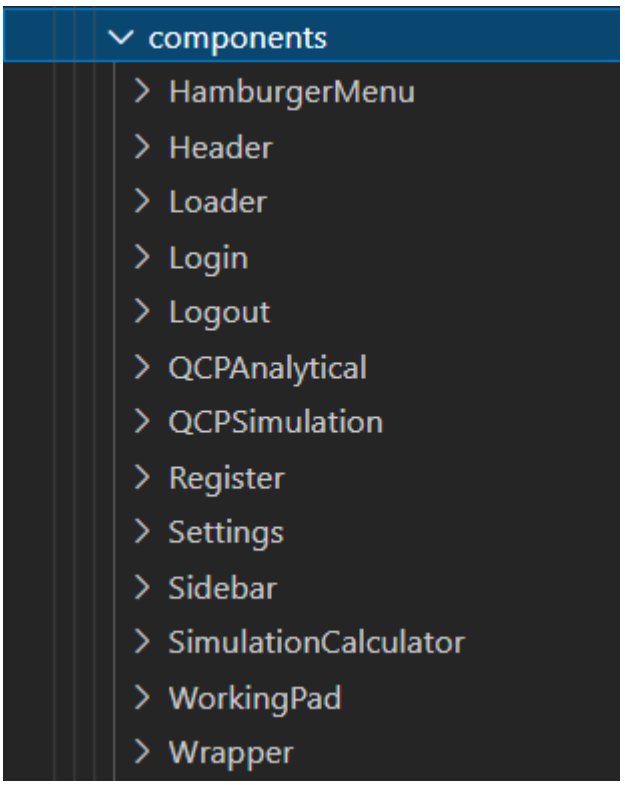


Fig: 14

1. HamburgerMenu: Component that shows and contains code for a hamburger menu and can be used whenever required by providing the list of options as a parameter..

2. Header: Component that contains code for the header.

3. Loader: Component that contains code for the loading screen.

4. Login: Component that displays login form and makes the API call on submission.

5. Logout: Component that deletes the user token and thus logs out the user.

6. QCPAnalytical: Component that contains code for the Quick-Calculator-Pad Analytical window.

7. QCPSimulation: Component that contains code for the Quick-Calculator-Pad Simulation window.
8. Register: Component that contains code for Register form and makes the API call on submission.

9. Sidebar: Component that contains code for Sidebar and takes the sidebar menu options as a parameter.

10. SimulationCalculator : Contains code for
    a. SimulationCalculator: Component that contains code for entering simulation details such as simulation time and number of simulations and makes an API call on submission.
    b. SimulationDetails: Component that contains code for displaying the data returned from the previous API call in a tabular format.

11. WorkingPad : Component that contains code for rendering Excalidraw Component in code as well as the sidebar menu options.

12. Wrapper : Component which takes true or false as a **props** parameter and displays child components accordingly.

5.5 Excalidraw

Excalidraw is a virtual collaborative whiteboard tool that lets you easily sketch diagrams that have a hand-drawn feel to them.
Link: https://excalidraw.com/

We have incorporated Excalidraw in our code (https://github.com/excalidraw/excalidraw) as it provides a drawing board and includes all basic functionalities such as dragging, dropping, zooming in and out, drawing lines and shapes etc. Upon manipulating the code and adding our own set of features we allow users to draw RBDs which can be simulated.

We have deployed our own version as an npm package and are using that.
(https://www.npmjs.com/package/@sohamkapileshwar/excalidraw)
What is an RBD? RBD or reliability block diagram is **a diagrammatic method of analysis used to assess the reliability of a complex system**.

To draw an RBD we need 2 basic elements: BLOCK and CONNECTOR.

Excalidraw by default provides the following drawing elements:

Fig: 15

Upon creating block and connector element, and removing the inessential elements we have 3 main elements:

Fig: 16

1. Selection: Element used to select objects. On selecting any element we get a popup window for customization of the element as shown below.

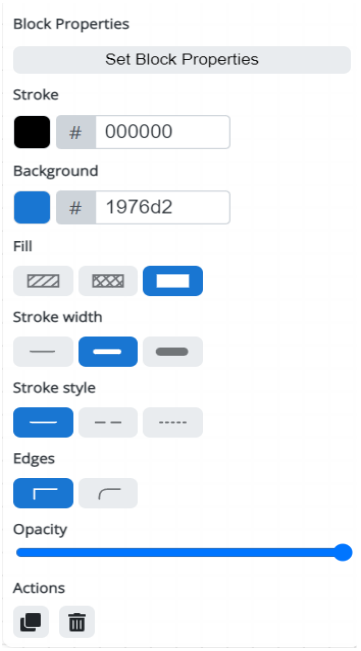


Fig: 17

2. Block: Rectangular-shaped object that has its own set of block properties and thus can act as a component in an RBD.

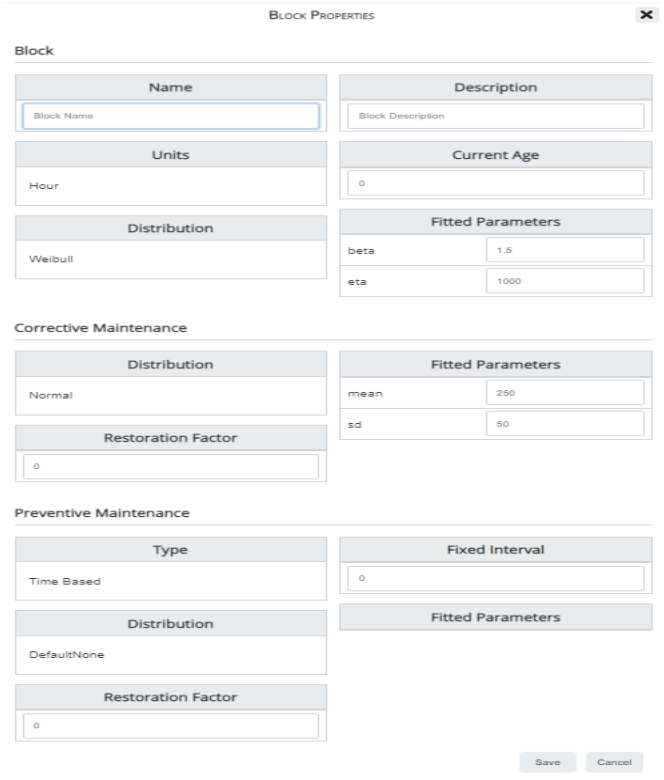


Fig: 18

3. Connector: Element used to connect or link two blocks thus allowing users to create various series / parallel and complex configurations.

Chapter 6 - Development of WebTool: Backend

6.1 Introduction

The data access layer of our software can be divided into two parts: Backend and Database.

The server-side of a website is referred to as the Backend. It ensures that everything on the client-side of the website functions properly. It is the section of the website that you are unable to view or interact with. It is the part of the software that has no direct touch with the users. Users have indirect access to the pieces and attributes created by backend designers via a front-end application.

A database is an information that is set up for easy access, management and updating.

PostgreSQL

PostgreSQL is a free and open-source Relational Database Management System(RDBMS). PostgreSQL database follows ACID properties before and after a transaction. In our project, we are using PostgreSQL as our database management system.

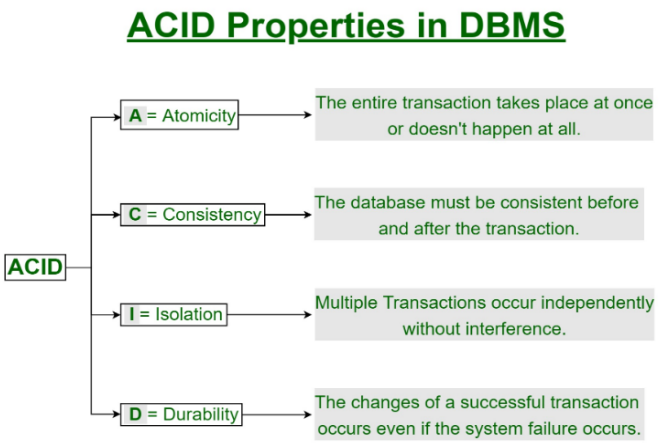


Fig: 19

Django REST framework

Django is a Python-based free and open source web framework. The Django REST framework (DRF) is a toolkit built on top of the Django web framework that reduces the amount of code you need to write to create REST interfaces. In our project, we are utilising this toolkit in order to create an API (Application programming interface).

REST API

An API is referred to as a contract between an information provider and an information user - establishing the data required from the consumer (the call) and the data provided by the producer (the response).

REST is an architectural constraint rather than a protocol or standard. REST may be implemented in a variety of ways by API developers. Our project involves the creation of a REST API in order to communicate with the data stored in the database.

6.2 Database Structure

6.2.1. Tables

In SQL, a table is a database object made up of rows and columns. To put it another way, it's a collection of connected data stored in a tabular format. Rows are referred to as records and the Columns are referred to as fields. A column is a collection of data values of a certain type (such as integers or alphabets), one value for each row of the database, such as Age, Student ID, or Student Name. A row in a table represents a single data item, and each row in the table has the same structure.

6.2.2. Table fields and types

Every Column in a table has a particular Data type. These data types depend on the DBMS we are using. Below are a few field types for PostgreSQL.

- **Boolean** - *true*, *false* and *null* are the possible values.
- **Characters** - Three character data types exist in PostgreSQL. CHAR(n), VARCHAR(n), and TEXT. CHAR(n) is used for data(string) having fixed-length of characters containing spaces. TEXT is variable-length character string. It can store unlimited length data. VARCHAR(n) has variable-length character string.
- **Numeric** - Two types of numbers exist in PostgreSQL, integers and floating-point numbers.
- **Arrays** - An array column in PostgreSQL may be used to hold an array of characters, an array of numbers, and so on.

6.2.3 Table Relationships

In a relational database, there are three types of table relationships. By specifying the appropriate foreign key constraints on the columns, the relationships may be enforced.

One-to-One

A record in one table is related to one record in another table. For example, One person has one ID number, and the ID number is unique to one person.

One-to-Many

In a database management system, a one-to-many relationship is a relationship between instances of one entity and several instances of another entity.
A student, for example, can work on many projects. Here, students and projects are separate entities. A One-to-Many relationship would be a single student working on two projects at the same time.

Many-to-Many

Multiple records in one table are related to multiple records in another table. For example, let’s say we are creating a database for a university. A student can be enrolled in multiple classes at a time and a class can have many students.

6.2.4 Entity Relationship Diagram

An entity relationship diagram (ERD) depicts the relationships between entity sets in a database. In this sense, an entity is an object, a data component. An entity set is a group of entities that are comparable in some way. Attributes can be assigned to these entities to specify their properties.
An ER diagram depicts the logical structure of databases by specifying the entities, their properties, and the interactions between them. ER diagrams are used to sketch out a database's design.

Below is the complete ER Diagram for our project.

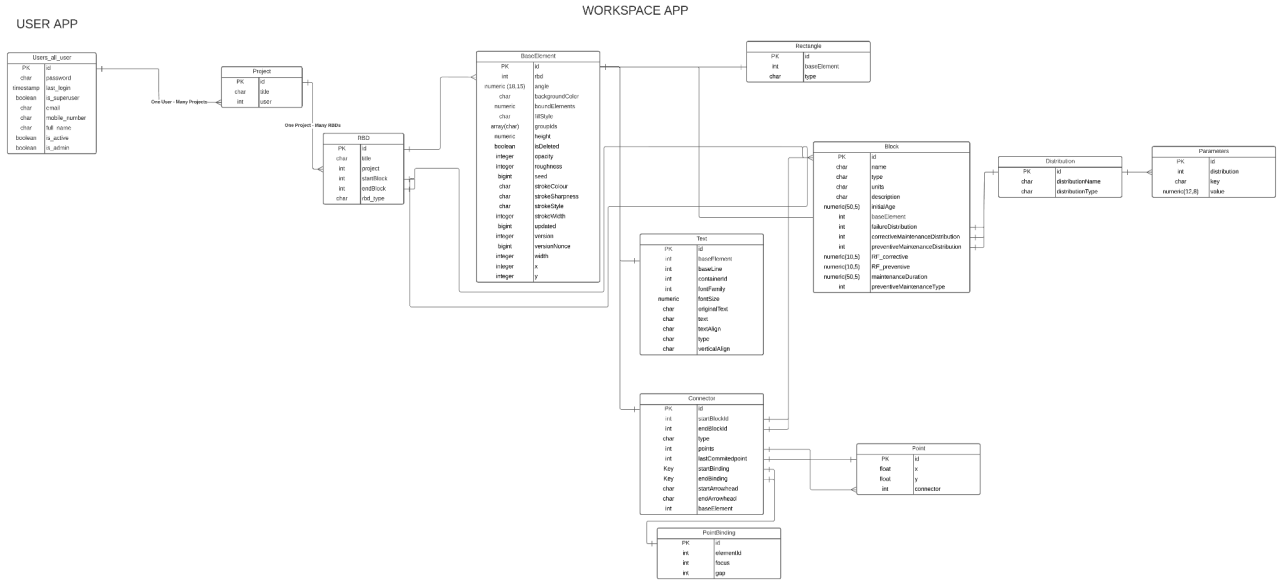


Fig: 20

6.3 Backend Structure

Our project utilises Django and Django REST framework in order to create the Backend for the Cloud-based Web-Tool. The Django REST framework (DRF) is an open source, mature, and well-supported Python/Django toolkit for creating complex web APIs.

6.3.1. Django App

A Django app is a small library representing a discrete part of a larger project. Our Backend consists of two Apps:

1. **Users App -** The Users App consists of all the code related to managing the users for the application. It handles API calls related to registering, logging in, logging out users etc.
2. **Workspace App -** The Workspace App consists of all the code related to managing other parts of the application like creating and saving Project and RBD, carrying out a simulation and returning the results etc.

6.3.2. Models

A model is the one and only source of knowledge about your data. It includes the most important fields and actions of the data you're saving. In general, each model corresponds to a single database table.

Users App Models

1. **All_User**: It defines the structure for the User Table. Contains fields that store email, full name, mobile number etc. It uses the MyUserManager class to override the creation of user and superuser.

Workspace App Models

1. **Project**: Defines the Project structure.
2. **RBD**: Defines the structure for a Block Diagram. It contains the startBlock and endBlock field which tells the starting block and ending block for the RBD.
3. **BaseElement**: BaseElement defines the structure for a basic element in a RBD. It contains the fields which are common across all elements.
4. **Block**: Block Element defines the attributes for a component. A component contains a failure distribution, maintenance distribution, repair distribution etc.
5. **Distribution**: Distribution Model contains a distribution name and type.
6. **Parameters**: Parameter Model contains a key and a value field. It's a generic model for creating parameters for different distributions.
7. **Connector**: Connector Element defines the attributes for a connector component. Connector is the joining arrow between Block Elements.

6.3.3. Views

Views are Python functions or classes that receive a web request and deliver a web response in the Django framework. The response might be a plain HTTP response, an HTML template response, or an HTTP redirect response that sends the user to a different website.

User App Views

1. **UserViewset**: Contains functions for creating a new user and listing user details.
2. **logout_view**: Logs the user out by deleting the Token from the database.

Workspace App Views

1. **ProjectViewset**: Contains functions to create a new project and list all the projects of a user when requested.
2. **RBDViewset**: Contains functions to create a RBD within a project and list the RBD's when provided with the project ID.
3. **Elements**: The Elements View accepts a GET and POST requests. GET request returns a response with all the elements in the RBD, app state and rbd_type. POST request handles the creation, updating and deletion of elements in the RBD.

6.3.4 Example working of an API call

The majority of web APIs to exist between the application and the web server. The user makes an API request instructing the programme to do something, and the application then uses an API to instruct the webserver to do something. The API serves as a bridge between the application and the web server, and the API call serves as the request.

Let's consider the API call for registering a new user on our platform. When a user enters his details and clicks on Register, the frontend makes a POST request to "`{Base_Url_Api}/user/AllUser/`". It sends a request body which contains the name, mobile number, password and email. This request is then handled by the UserViewset in the backend which creates a new User in the database and returns a response which looks like this,


```
{
    "email": "sohamtest@gmail.com",
    "mobile_number": "9869684520",
    "full_name": "Soham Kapileshwar",
    "token": "414a7728b6599443d710e554dcf9c6b0d82427a5"
}
```


Fig: 21

6.4 API Endpoints Documentation

Here is a list of APIs and their function. All URLs by default have the Base Url at the start. The list of URLs can be found in frontend/api/Urls.js. Each API Endpoint, except register, takes two header attributes,

1. Authorization - Token ${user token}
2. Content-type - application/JSON

Users App

1. **LOGIN**

POST	/user/login
Logs in the user if the credentials provided are correct.	
REQUEST BODY username: *String* password: *String*	**RESPONSE BODY** token: *String*

2. **LOGOUT**

GET	/user/logout
Logs out the currently logged in user.	
REQUEST BODY	**RESPONSE BODY** response: *String*

3. **ALL USER**

GET	/user/AllUser/
Retrieves the user details of the current user.	
REQUEST BODY	**RESPONSE BODY** id: *number* email: *String* mobile_number: *String* full_name: *String*

POST	/user/AllUser/
Registers a new user.	

REQUEST BODY email: *String* mobile_number: *String* full_name: *String* password: *String* password2: *String*	**RESPONSE BODY** id: *number* email: *String* mobile_number: *String* full_name: *String* token: *String*

Workspace App

1. **PROJECT**

GET	/workspace/Project/
Retrieves all the projects for the current user.	
REQUEST BODY	**RESPONSE BODY (List)** id: *number* title: *String* user: *number*

POST	/workspace/Project/
Creates a new project.	
REQUEST BODY title: *String*	**RESPONSE BODY** id: *number* title: *String* user: *number*

PUT	/workspace/Project/${project_id}/
Edits the title of the project.	
REQUEST BODY title: *String*	**RESPONSE BODY** id: *number* title: *String* user: *number*

DELETE	/workspace/Project/${project_id}/
Deletes the selected project.	
REQUEST BODY	**RESPONSE BODY**

2. **RBD**

GET	/workspace/RBD/?project=${project_id}/
Retrieves all the RBDs for the current project.	
REQUEST BODY	**RESPONSE BODY (List)** id: *number* title: *String* project: *number*

POST	/workspace/RBD/
Creates a new RBD inside the current project.	
REQUEST BODY title: *String* project: *number*	**RESPONSE BODY** id: *number* title: *String* project: *number*

PUT	/workspace/RBD/${rbd_id}/
Edits the title of the RBD.	
REQUEST BODY title: *String*	**RESPONSE BODY** id: *number* title: *String* project: *number*

DELETE	/workspace/RBD/${rbd_id}/
Deletes the selected RBD.	
REQUEST BODY	**RESPONSE BODY**

3. **ELEMENTS**

GET	/workspace/Elements/?RBD={rbd_id}/
Retrieves all the elements for the current RBD.	
REQUEST BODY	**RESPONSE BODY (List)** elements: *Array of element object* rbd_type: *String* appstate: *{* *startBlock: element object*

	endBlock: element object *}*

POST	/workspace/Elements/?RBD=${rbd_id}/
Creates, updates and deletes elements for the current RBD.	
REQUEST BODY elements: *Array of element object* rbd_type: *String* *startBlock: element object* *endBlock: element object*	**RESPONSE BODY** elements: *Array of element object* rbd_type: *String* appstate: *{* *startBlock: element object* *endBlock: element object* *}*

4. **QUICK CALCULATION PAD**

POST	/workspace/QCP/?calculate=${parameter}/
Calculates and returns the parameter specified.	
REQUEST BODY distributionName: *String* parameters: *Object* time: *String* start_time: *String* required_reliability: *String* X: *String*	**RESPONSE BODY** QCP_result: *String*

5. **CHECK IF RBD IS VALID**

GET	/workspace/checkValidRBD/?RBD=${rbd_id}/
Checks if the RBD specified is valid before performing simulation and returns the simulation function to be used from Fsim.	
REQUEST BODY	**RESPONSE BODY** status: *String* response: *String* function: *String*

Another API Endpoint has been created which handles the simulation of a RBD. Working of this API call is explained in detail in the next chapter.

6.5 Simulation and Fsim Library

Our project utilizes the Fsim Library in order to perform simulation. “Fsim” is a Python 3 failure simulation package that offers programs, in the form of functions, for all fundamental maintenance techniques and system configurations. This library allows us to generate simulation results for a given system setup under specified conditions with a single line of code.

On clicking the Simulate button on our UI an API call is made to the corresponding endpoint,

POST	/workspace/Simulate/?RBD=${rbd_id}/
Simulates the RBD for the specified time and iterations and returns the result.	
REQUEST BODY simulation_time: *number* number_of_simulations: *number* function: *String*	**RESPONSE BODY** *Fsim Solution Object*

This API Endpoint runs the “Simulate” function, which calls the “SimulationWrapper” in utils.py.

SimulationWrapper performs the following function in the specified order:

1. Gets all the Elements for the RBD to be simulated.
2. Checks which function is to be called in Fsim Library and sends the required arguments accordingly to the Fsim Function.
3. Serializes and returns the result.

Currently our project incorporates three cases for simulation - Single component, Series Configuration, Parallel Configuration.

The result returned is of type *Fsim Solution Object,* which is a defined data structure and can be found in Fsim folder defined as class Solution. It contains specific attributes depending on the function called for performing simulation. Some attributes are as follows:

sys_av: System Availability (Para Object)

sys_DT: System Downtime (Para Object)

sys_UT: System Uptime (Para Object)

sys_NODT: System Number of Downtimes (Para Object)

sys_NOCM: System Number of Corrective Maintenances (Para Object)

sys_NOPM: System Number of Preventive Maintenances (Para Object)

comp_NOCM: Component Number of Corrective Maintenances (List)

comp_NOPM: Component Number of Preventive Maintenances (List)

A Para Object is also a defined data structure which contains,

array: List of individual simulation results (List)

mean: Mean value of all results (Float)

std: Standard deviation of all results (Float)

min: Minimum of all results (Float)

max: Maximum of all results (Float)

The returned result is then displayed on the Web-Tool in a tabular format.

Chapter 7: Demonstration of the WebTool using a sample problem

This chapter would be a demonstration of the Cloud-Based WebTool in order to simulate a system and get the necessary results.

Consider a system which is comprised of four serially related components each having a Weibull time to failure distribution with parameters as shown in the table,

Component	Scale parameter	Shape parameter
1	100	1.20
2	150	0.87
3	510	1.80
4	720	1.00

Each component follows a Normal Distribution for corrective maintenance with a mean of 250 and a standard deviation of 50.

We need to simulate this given system for 5000 hours and 10 iterations and estimate the system availability, number of corrective maintenances that might be required and other important parameters.

This problem can be easily and interactively simulated on our Cloud-Based WebTool. This is what the step by step process for the same would look like,

1. Login/Register on the application by entering valid credentials.
2. Create a new project and within the project create a new RBD(Reliability Block Diagram).

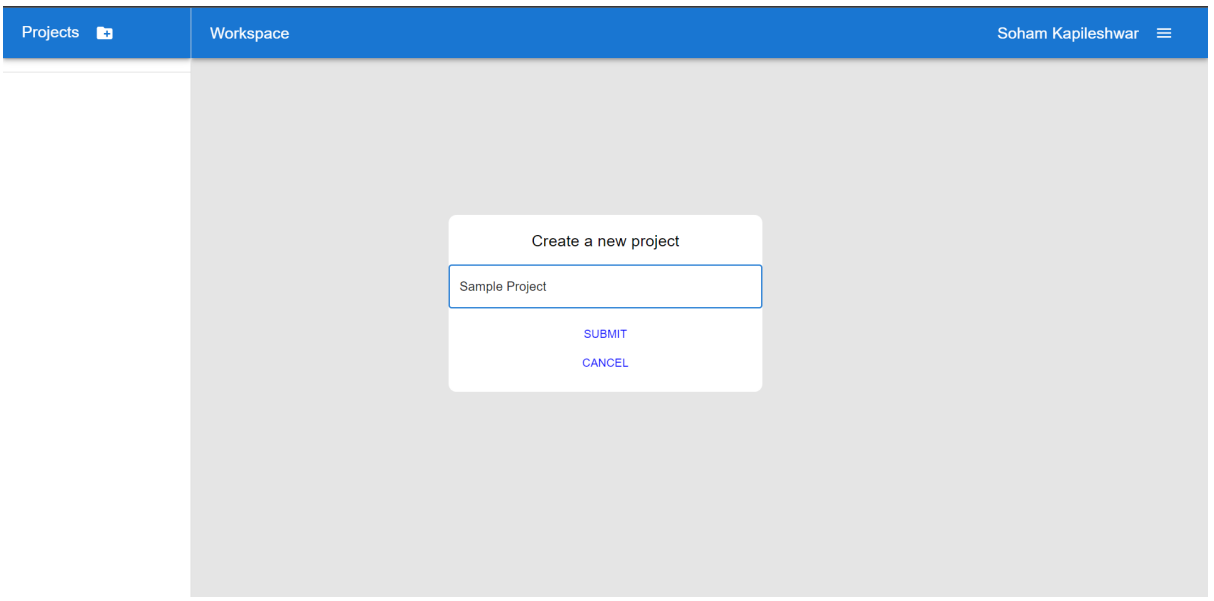


Fig: 22

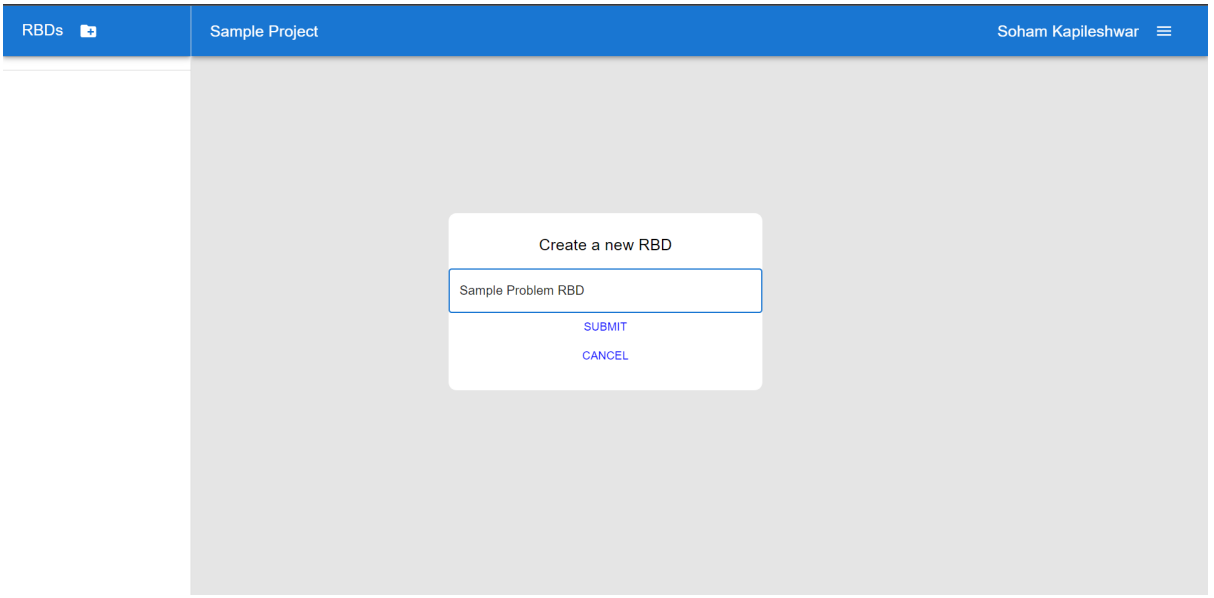


Fig: 23

3. Start creating the series configuration by using the “Block” and “Connector” element on the top toolbar. Create four blocks and connect them in series.

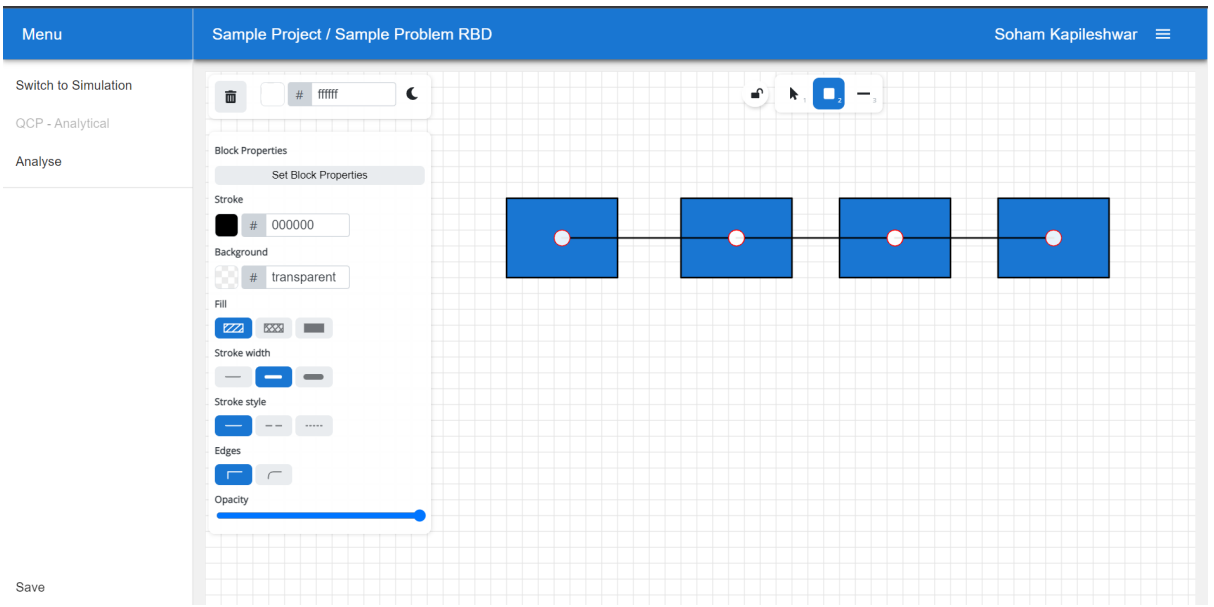


Fig: 24

4. Click on each block and go to “Set Block Properties”. Assign Weibull Failure Distribution to each and enter the corresponding scale and shape parameters. Assign Normal Distribution for corrective maintenance and enter the mean and sd.

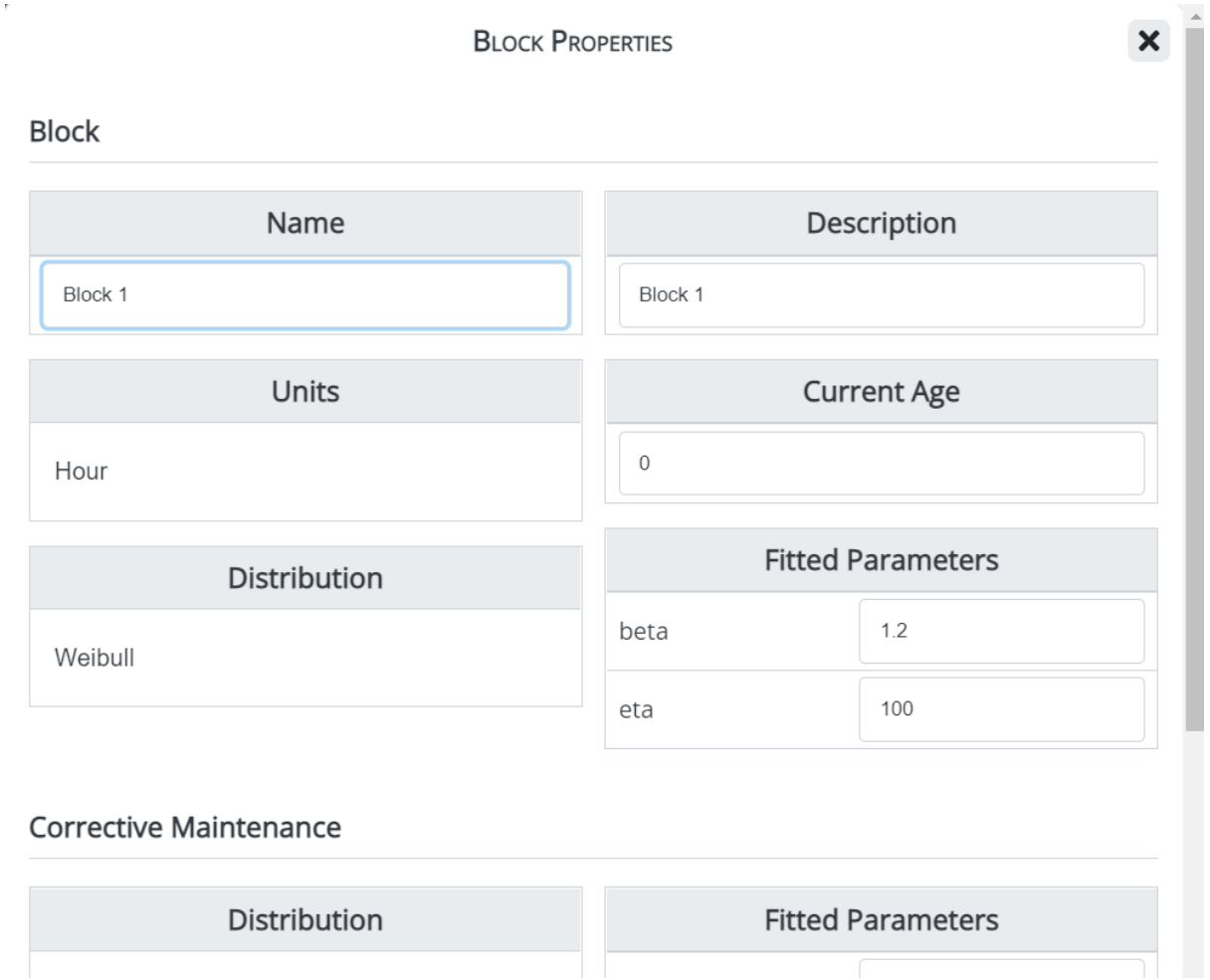


Fig: 25

5. Right click the first component and set it as the starting block. Similarly right click the last component and set it as the ending block.
6. Click on Save on bottom left and “switch to simulation” in order to perform a simulation.
7. Click on the “Simulate” button and enter the simulation time and number of iterations. Then click on simulate in order to perform the simulation.

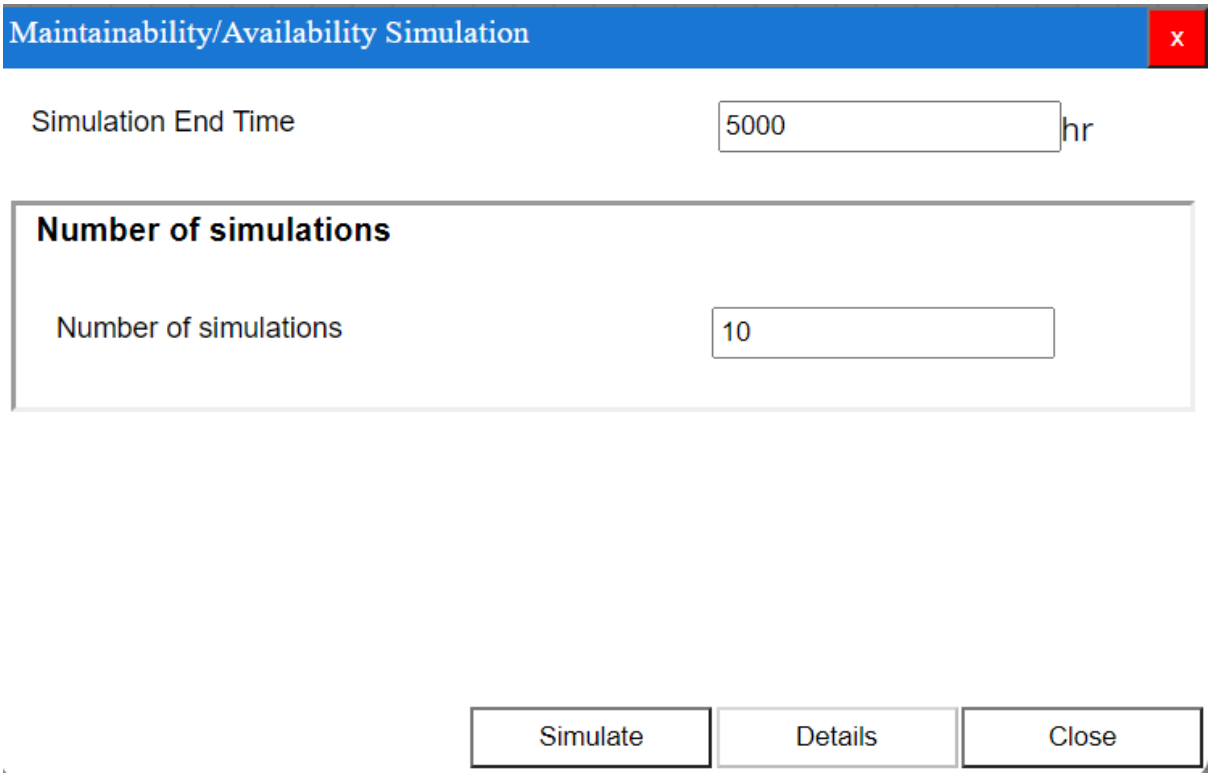


Fig: 26

8. The Details button would turn active once the simulation is completed. Click on the same to get the results.

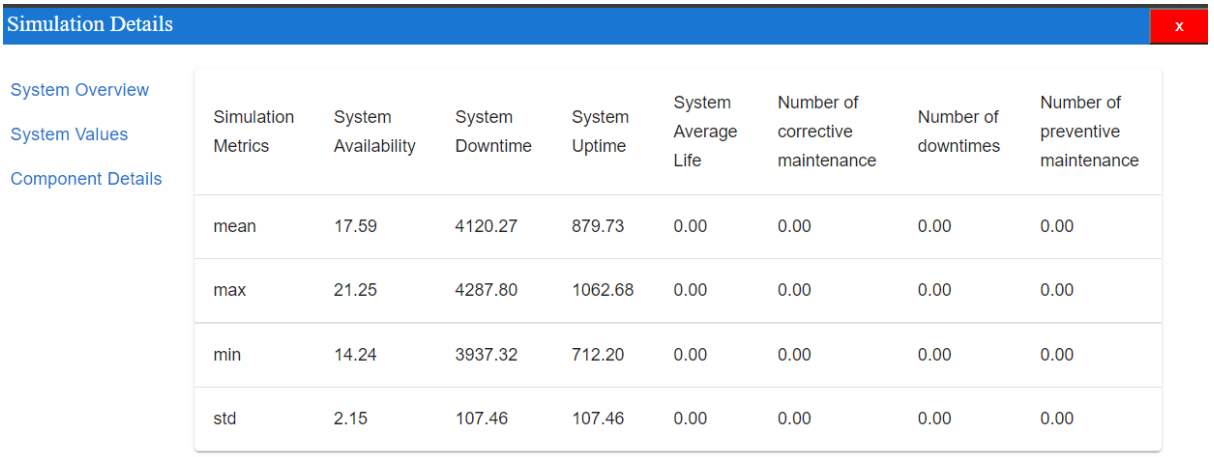


Fig: 27

Chapter 8: Results & Conclusion

8.1 Results

On comparing the analytical and simulation results with BlockSIM software, there was an error margin varying from 1% to 5% suggesting that the results are accurate and the software can be used for analysis.

Simulation Metrics	System Availability	System Downtime	System Uptime
mean	18.37	4081.74	918.26
max	22.91	4300.24	1145.50
min	14.00	3854.50	699.76
std	2.26	113.03	113.03

Number of corrective maintenance	Number of corrective maintenance	Number of corrective maintenance	Number of corrective maintenance
1.20	1.50	6.10	10.30

Fig: 28

System Overview	
General	
Mean Availability (All Events):	0.184676
Std Deviation (Mean Availability):	0.022626
Mean Availability (w/o PM, OC & Inspection):	0.184676
Point Availability (All Events) at 5000 hr:	0.1
Reliability at 5000 hr:	0
Uptime (hr):	923.380601
Total Downtime (hr):	4076.619399

Number of CMs
10.2
5.8
1.6
1.1

Fig: 29

8.2 Conclusion

Thus the **cloud-based tool for reliability estimation** is an effective and economical solution for SMEs (small to medium enterprises) or individuals looking to perform reliability analysis for a short-term or infrequent period. Users are simply required to create an account, and then have access to create an RBD, perform analysis and save their progress to the cloud. Thus only an internet connection is required to run the web tool.

Though there are various softwares available in the market which can give more detailed and sophisticated results of simulations compared to this web tool , the web-tool has benefits of its own.

As it is a cloud-based application, the user requires only an internet connection to access data and thus removes the need for installing it in your device (softwares like BlockSIM). Being online data can be accessed from multiple devices. Currently, without a subscription model , it is free-of-cost to use and thus removes the heavy incurring charges of softwares like BlockSIM.

8.3 Future Scope

8.3.1 Features and Scalability

Although capable of executing all types of basic simulation and analytical scenarios for most basic maintenance plans and system configurations, this web-tool is still in its early stages and has the potential to evolve in every direction.

The future scope of the web-tool is to further develop and to include the following features :-

1. Plotting more graphs on various iterations of parameters giving a broader analysis.
2. More complex configurations of RBD including k out of m, series-parallel combination etc. Currently, only series and parallel configurations can be analysed.
3. Improving the accuracy of the simulation results.
4. Hosting the application on a better cloud service such as AWS.
5. Using graph algorithm for simulation (8.2.2)

8.3.2 Subscription model

Softwares like BlockSIM have a subscription model that incur heavy charges for a long period of usage. Thus for users in need of short- term or infrequent usage, it is not feasible. Introducing a "**pay-per-use**" model is a viable solution as users can only pay for their limited usage of features and not the entire software for a longer period.

8.3.3 Graph Algorithm for simulation

Currently our project uses the Fsim Library for performing the simulation. The way Fsim Library works is it categorizes the system into a particular configuration. These categories are basically the various functions created in Fsim Solution.py file. For example, the function "Block_series_TABM" is used when we have n blocks in series which undergo time or age based preventive maintenance. "Block_parallel_TABM" is used when we have n blocks in parallel which undergo time or age based preventive maintenance.

The limitation of such an approach is that there is no generalized way to simulate any kind of configuration. In reality, systems can be really complex and this approach might not have a category for the same.
A better and more efficient approach is to utilize Graph Data Structure to solve this problem. Here is a glimpse of how this method would work.

Consider the user creates any kind of complicated configuration. The Algorithm would create a graph, where the blocks would be the **nodes** and the connectors would be the **edges** of the graph. In order to simulate the system, Monte Carlo Simulation method can be used. If a block is undergoing repair, that node can be disconnected from the graph. In order to figure out if the system is working or not, a graph traversal can be done from the starting node to the ending node.
The biggest advantage of this approach is that it would be a single algorithm to perform simulation on any kind of system, instead of multiple categories and functions.

References

[1] An Introduction to Reliability and Maintainability Engineering by Charles E. Ebeling.

[2] Raychaudhuri, Samik. "Introduction to monte carlo simulation." *2008 Winter simulation conference*. IEEE, 2008.

[3] Chanan Singh; Panida Jirutitijaroen; Joydeep Mitra, "Analytical Methods in Reliability Analysis," in *Electric Power Grid Reliability Evaluation: Models and Methods* , IEEE, 2019, pp.117-164, doi: 10.1002/9781119536772.ch5.

[4] Development of a Failure Simulation Library Based on Concepts of Reliability Engineering by Chinmay Naik.

[5] BlockSim/RENO 11 User's Guide - Reliasoft.

[6] Xie, Ming, and Chin Diew Lai. "Reliability analysis using an additive Weibull model with bathtub-shaped failure rate function." *Reliability Engineering & System Safety* 52.1 (1996): 87-93.

[7] Hallinan Jr, Arthur J. "A review of the Weibull distribution." *Journal of Quality Technology* 25.2 (1993): 85-93.

[8] Breneman, James E., Chittaranjan Sahay, and Elmer E. Lewis. *Introduction to reliability engineering*. John Wiley & Sons, 2022.